ILLINOIS INSTITUTE OF TECHNOLOGY

Application of DMNetwork and TS in Transient Stability Simulator for Power Systems

CS 595: Advanced Scientific Computing
Final Project

by

Bikiran Guha and Jianqiao Huang

12/8/2016

Table of Contents

Abstract 2
Introduction 3
Base Power System Model 4
DMNetwork and TS 6
Code Implementation 7
Results & Analysis 10
Conclusion and Future Work 11
References 11

Abstract

This project has used DMNetwork (a class in PETSc) to organize a 9 bus power system consisting of 3 generators, 3 loads and 9 transmission lines. The project uses TS to simulate the transient stability of the network in response to a short circuit fault at one of the buses. The simulations can be executed using multiple processors. The program was then extended to be able to create user-defined number of copies of the 9 bus power system model and physically connect them together radially. In this manner, a 90,000 bus system (having 450,000 variables) was created and scalability of the system was tested using multiple processors.

Introduction

The electric power system can be broadly categorized into generation, transmission and distribution. Generators are the power source, i.e., they generate electrical energy from various fuel sources like coal, natural gas, nuclear, hydro, wind or sunlight. The electrical energy is then transported over long distances via transmission lines to the distribution stations, where it is finally distributed to the consumers. Figure 1 illustrates the basic structure of the power system

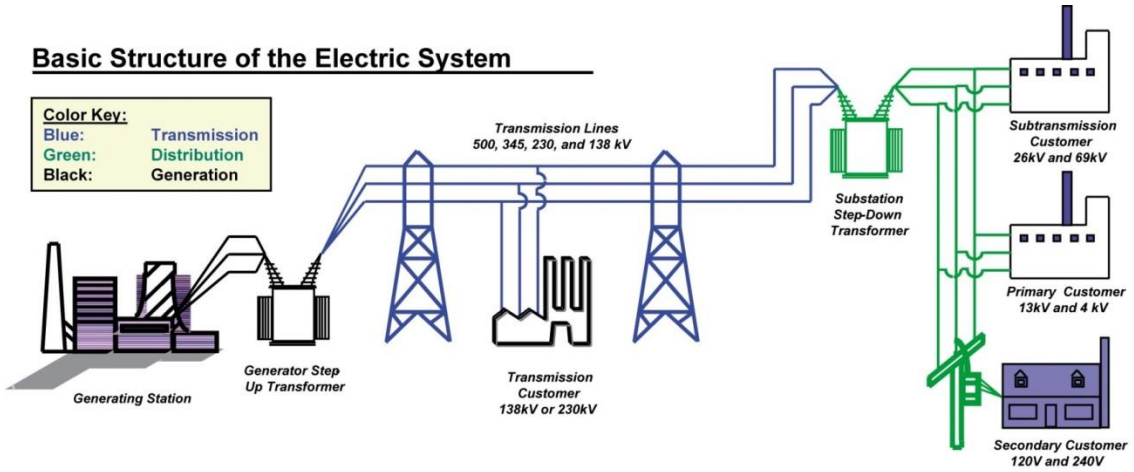

Figure 1 Subsystems in a power system

The transmission system plays an important role in the reliability of the system since it transports enormous amount of power from the generation subsystem to the load centers. Therefore, the utility operators need to take preventive actions so that the continued operation of the transmission system can be ensured in case of severe disturbances such as short circuit faults. Transient stability analysis is one such important preventive action.

Transient stability simulators for power systems are used to visualize the electromechanical interactions of the generators within the power system. These simulators help determine the stability of the power system following a transient such as a fault or a load switching event. By stability, we mean the ability of the system to return to a normal operating state after a transient event. The stability of the system in reaction to an event depends on the current operating state, the magnitude of the event, the protection mechanism in place and preventive or corrective actions taken.

In this project, the transient stability of a small power system is tested in response to a short circuit fault using the PETSc library and its features such as DMNetwork and TS. The simulation model of this small power system is identical to an example in the PETSc library (ex9bus), but that example does not use DMNetwork and can only run with a single processor. The simulation results have been matched with that of ex9bus (both in sequential and parallel) to ensure accuracy.

The organizing features of DMNetwork have been utilized in the project to create a program which can accept input from the user to create a certain number of copies of the base power system. The program then radially connects the copies to create a larger power system. In this way, the user can create a very large power system having a bus number which is a multiple of 9 and can test the stability of the system in response to faults at various locations in the system.

Finally, in this report, the scalability of parallel processing has been demonstrated in a 90,000 bus system having 450,00 variables.

Base Power System Model

The power system model has been shown in Figure 2. It consists of 9 buses (electrical nodes), 3 generators, 3 loads and 9 transmission lines. The model is representative of a simple transmission system.

The equations in the model are differential-algebraic in nature, i.e., there are some equations which are differential while others are algebraic. The equations can be divided into two subsystems: **Network subsystem** and **Generator subsystem**.

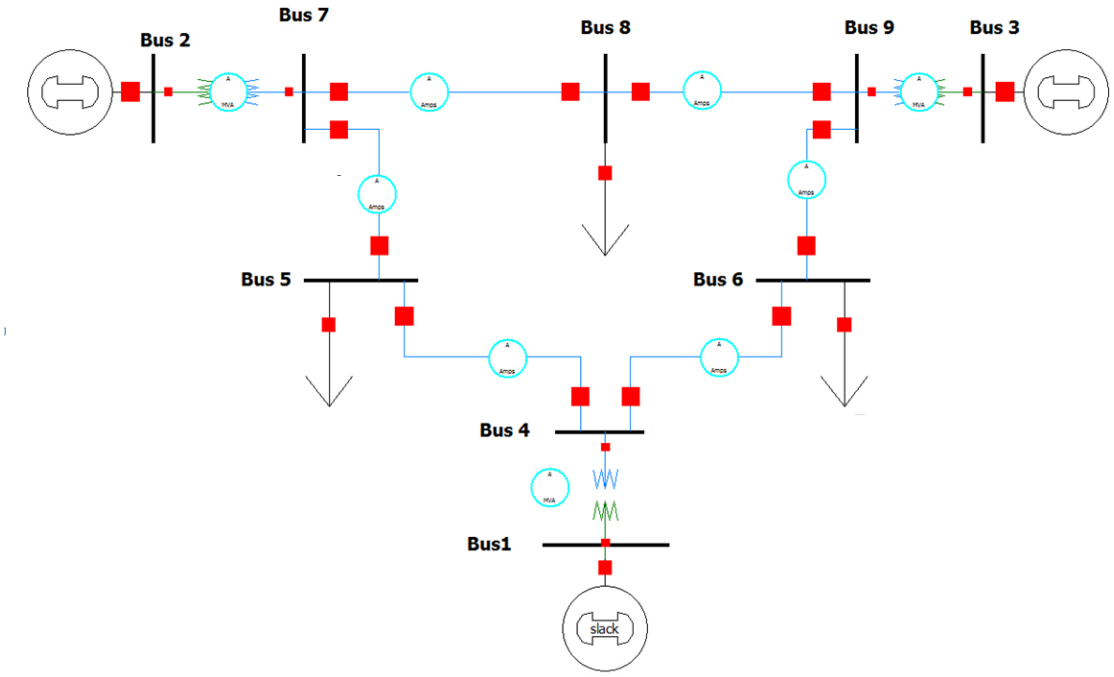

Figure 2 Model for 9 bus power system

The **network subsystem** consists of the algebraic equations associated with the network. These equations represent the current balance in the system, i.e., the algebraic sum of

currents meeting at a bus has to be zero. Since the current is complex (has real and imaginary components), there are 2 current balance equations for each bus:

$$(\boldsymbol{I_L})_{\boldsymbol{real}} + (\boldsymbol{I_G})_{\boldsymbol{real}} + (\boldsymbol{I_{branch}})_{\boldsymbol{real}} = \mathbf{0}$$
$$(\boldsymbol{I_L})_{\boldsymbol{imag}} + (\boldsymbol{I_G})_{\boldsymbol{imag}} + (\boldsymbol{I_{branch}})_{\boldsymbol{imag}} = \mathbf{0}$$

where $(\boldsymbol{I_L})_{\boldsymbol{real}}$, $(\boldsymbol{I_G})_{\boldsymbol{real}}$ and $(\boldsymbol{I_{branch}})_{\boldsymbol{real}}$ are the real components of the load current, generator current and branch currents respectively and $(\boldsymbol{I_L})_{\boldsymbol{imag}}$, $(\boldsymbol{I_G})_{\boldsymbol{imag}}$ and $(\boldsymbol{I_{branch}})_{\boldsymbol{imag}}$ are the imaginary components of the load current, generator current and branch currents respectively.

The **generator subsystem** consists of 9 differential-algebraic equations associated with the generator and its exciter. These equations model the dynamics of the generator subsystem. The equations are provided below and are from [1] which contains a detailed description of generator subsystem models.

$$\frac{d\delta_i}{dt} = \omega_i - \omega_s$$
$$T'_{qoi}\frac{dE'_{qi}}{dt} = -E'_{di} + \left(X_{qi} - X'_{qi}\right)I_{qi}$$
$$\frac{2H_i}{\omega_s}\frac{d\omega_i}{dt} = T_{Mi} - E'_{di}I_{di} - \left(X'_{qi} - X'_{di}\right)I_{di}I_{qi} - D_i(\omega_i - \omega_s)$$
$$T'_{doi}\frac{dE'_{di}}{dt} = -E'_{qi} - (X_{di} - X'_{di})I_{di} + E_{fdi}$$
$$T_{Ei}\frac{dE_{fdi}}{dt} = -\left(K_{Ei} + S_{Ei}\left(E_{fdi}\right)\right)E_{fdi} + V_{Ri}$$
$$T_{Fi}\frac{dR_{fi}}{dt} = -R_{fi} + \frac{K_{Fi}}{T_{Fi}}E_{fdi}$$
$$T_{Ai}\frac{dV_{Ri}}{dt} = -V_{Ri} + K_{Ai}R_{fi} - \frac{K_{Ai}K_{Fi}}{T_{Fi}}E_{fdi} + K_{Ai}(V_{refi} - V_i)$$
$$\left[Z_{d-q,i}\right]\begin{bmatrix} I_{di} \\ I_{qi} \end{bmatrix} = \begin{bmatrix} E'_{di} - V_{di} \\ E'_{qi} - V_{qi} \end{bmatrix}$$

Since there are 9 buses and 3 generators, there are a total of 45 variables in this simple power system model.

DMNetwork and TS

PETSC DMNetwork [2] is a class for managing and organizing general unstructured networks and therefore it is suitable for power grid applications. It is built on top of DMPlex, an object used to represent unstructured grids. The design elements in the DMNetwork consist of:

- Vertex
- Edge
- Components

Vertices represent the junctions or nodes in the network while the edge represent the connections between vertices. The components are the structures which contain the physical parameters of the different devices in the model.

Advantages of using DMNetwork:

- Vertices and edges can be added or removed easily
- DMNetwork has functions to keep track of variable indexes
- DMNetworkDistribute() can easily set up and partition the network
- Computations using PETSc solvers take place seamlessly over the network

In the project, for the base 9 bus system model, there are 9 vertices (1 for each bus), 9 edges (1 for each branch) and 4 components: bus, generator, load and branch.

PETSC TS is a library dedicated to the scalable solution of ordinary differential equations (ODEs) and differential algebraic equations (DAEs) arising from the discretization of partial differential equations (PDEs) [3]. With the help of the TS library, it was possible to evaluate the system's state variables at each time step and thereby get the response of the system during and after a transient event, such as a short circuit fault.

Code Implementation

The code for the developed simulator is written in C using the PETSc library framework and compiled with the GNU's gcc compiler with the gdb debugger. The source code is stored in a git repository. The steps of the project are summarized in Figure 3.

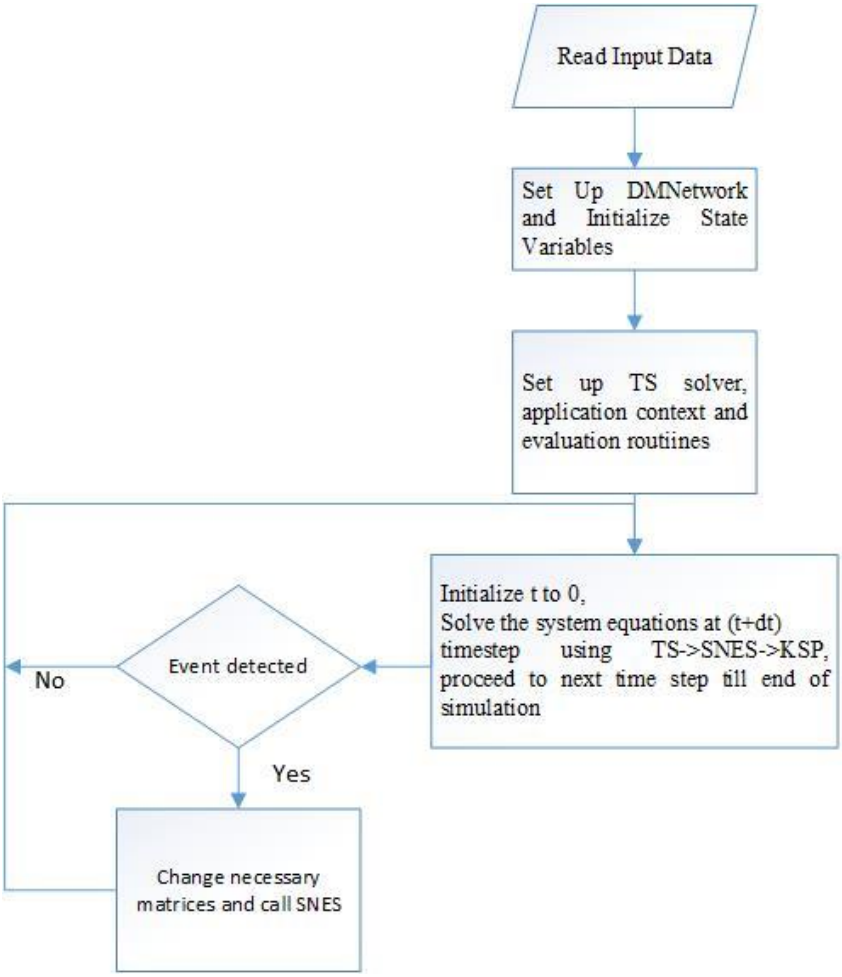

Figure 3: Project Flowchart

The first step of the program is to read the inputs to the model. If there are multiple processors, only the processor with rank = 0 is involved in the reading process. The binary files 'X.bin' and 'Ybus.bin' contain the initial values of the state variables (X) and the Ybus matrix. The Ybus matrix is used to calculate the branch currents ($I_{branch)}$ from the bus voltages (V_{bus}) from the following equation:

$$I_{branch} = Y_{bus} V_{bus}$$

where V_{bus} is the state variable vector for the network equations. Its initial value has been provided in the X.bin file. The values of X.bin are read into V0 vector and that of Ybus.bin are stored into the Ybus matrix. The function **read_data()** is used to read all the

inputs and store them inside data structures representing buses, generators, loads and branches. These data structures contain parametric data of the corresponding devices. **read_data()** also defines an edgelist which contains information about how the vertices are connected.

The next step involves creating the DMNetwork object using **DMNetworkCreate()** and specifying the number of vertices and edges using **DMNetworkSetSizes**(). It is important to mention here that, for parallel programming, only the rank 0 processor is involved in the DMNetwork setup. The network data structures are registered as components and given component keys, which is an identifier for the component. The bus, generator and load components are then added to the vertices while the branch component is added to the edges. If the number of processors is more than one, **DMNetworkDistribute**() distributes the network and moves the associated data.

Then vectors X, Xdot and F are created to store the state variables, their derivatives and the residual function values. The state variables are then initialized using **SetInitialGuess()** with V0 as input. A user context was created to contain information about the fault, such as fault on/off times, location of fault, etc.

The next step was to set up the TS solver. After creating the TS object using **TSCreate()**, it was added to the DMNetwork using **TSSetDM()**. Then the various options used by TS are set, such as the time-stepping solver method, the start time, end time and the step time. For the project, TS uses the Backward Euler Method to solve for X.

The **FormIFunction()** gets X, Xdot and time at the current step and sets up the equations in the residual function F for TS to solve. The code passes **FormIFunction()** to **TSSetIFunction()** which tells TS to solve the following equation:

$$F\big(t, X(t), \dot{X(t)}\big) = 0$$

When **TSSolve()** is called, TS calls Scalable Nonlinear Equations Solver (SNES) to solve the residual function of the non-linear equations. SNES linearizes the non-linear residual functions using the Newton's method and then calls **KSPSolve()** to solve the linear equations using the Krylov Subspace methods. The equations involved in the solving process can be summarized as:

$$X_{n+1} - X_n - \frac{\Delta t}{2}\big(f + f(X_n)\big) = 0$$
$$g(X_{n+1}) = 0$$

During switching events (fault-on and fault-off), the program gets the SNES object from TS using **TSGetSNES()** and gets **AlgFunction()**. In **AlgFunction()**, the residual functions of all the differential equations are set to zero and the Y_{bus} values of the fault bus are changed to represent a short circuit fault. **AlgFunction()** is then used by **SNESolve()** to get the values of X.

The simulation options used are:

TS type: Backward Euler
SNES solver: Newton's Method
SNES Relative Tolerance: 1e-8
SNES Absolute Tolerance: 1e-8
KSP solver: GMRES
Sequential Preconditioner: LU
Parallel Preconditioner:
-pc_type bjacobi
-sub_pc_type lu
-sub_pc_factor_shift_type NONZERO

After successfully implementing the 9 bus model in DMNetwork, **the program was modified to make copies of the model.** This was achieved using for loops inside **read_data()** and making component[i + ncomp] and component[i] identical with respect to the data they hold. Here, i: component index, ncomp: No. of the component inside the 9 bus system. For example, ncomp = 9 for buses and ncomp = 3 for generators. The same concept was applied to edgelist, i.e., edgelist[i+nedgelist] = edgelist[i]. The user can define the number of copies to make of the 9 bus model while executing the program by appending with '-nc <no. of copies>'.

Finally, the copies were connected radially, i.e., a branch was added between the last bus of copy[i] and the first bus of copy[i+1] and the edgelist was updated accordingly. Besides modifying **read_data()**, the program also modifies the DMNetwork dimensions in the main program to accommodate all the copies.

Results & Analysis

The file 9bus_dm1_l2.c in the git repository contains the source code of the program. The file 'petscoptions' contains all the options. Initially, the number of copies is set to 1 and the results are verified with the ex9bus.c example in PETSc library. Figure 4.1 shows the voltage magnitude of fault bus in the nine bus system while Figure 4.2 shows the direct axis current (I_d) value of generator 1. The total simulation time is set to 5 seconds. The simulation scenario is the application of a three phase fault on bus 9 at t= 1 seconds, and clearing it at t= 1.2 seconds. The fault resistance was set to 0.0001 pu.

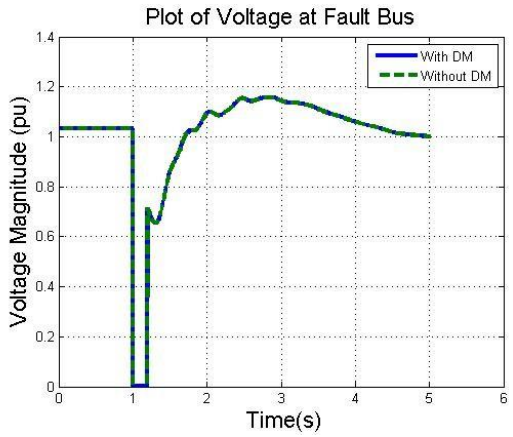

Figure 4.1 Voltage of bus 9 plots

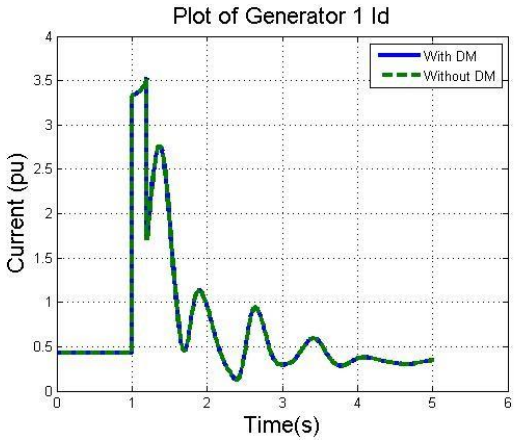

Figure 4.2 Generator 1 I_d current plots

The fault bus voltage drops to zero during the fault and then slowly recovers to the original value after the fault is cleared. The I_d current increases almost 7 times (from 0.5 p.u to 3.5 pu) during the fault. After the fault is cleared, the current values slowly return to normal. It can be seen from the two figures that the results match in both cases. Similarly, all the state variable values from 9bus_dm1_l2 match with those from ex9bus.

9bus_dm1_l2.c has also been tested on a ninety thousand bus system (nc = 10,000) with 450,000 variables. In this case, the three phase fault has been applied on bus 9 at t= 0.02 s, and the fault has been cleared at t= 0.05 s. The total simulation time is 1 s. The code has been tested with different number of processes to get the speed up and efficiency values in both debugging mode and optimization mode. Speed up (S_p) and Efficiency (E_p) are defined as:

$$S_p = \frac{\text{Uniprocessor ExecutionTime}}{\text{Parallel ExecutionTime}}$$

$$E_p = \frac{S_p}{p}$$

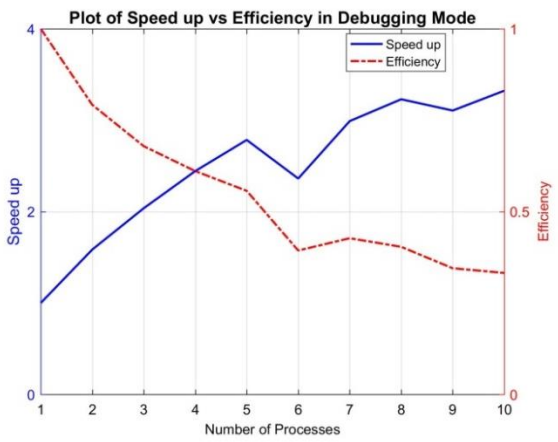

Figure 4.3(a) S_p vs E_p in debugging mode

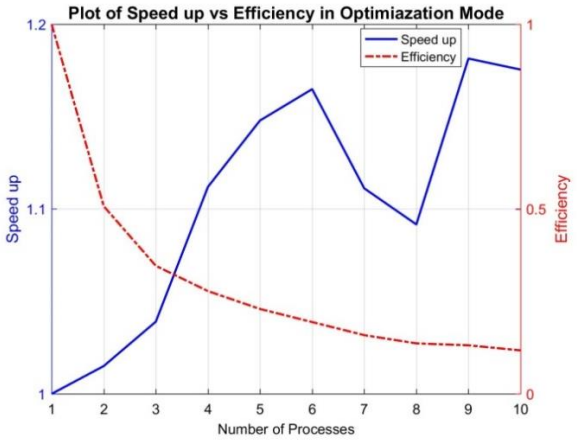

Figure 4.3(b) S_p vs E_p in optimization mode

Figure 4.3(a) and 4.3(b) show that, both in debugging and optimization mode, the speed up (S_p) increases as the number of decreases. However, the efficiency decreases as the number of processes increase. The efficiency decreases with increasing number of processors because of the increase in process waiting time with more number of processes. The equations to be solved are not independent of each other and as the number of processes increase, more and more processes need to wait for data from other processes to solve their own set of equations.

Conclusion and Future Work

This project involved the successful application of DMNetwork to organize a 9 bus power system and TS to analyze its transient stability in case of a short circuit fault. The results were verified with that of ex9bus example in PETSC library, which uses the same model. The program was further developed to be able to generate much larger power systems by making radially connected copies of the 9 bus case. A 90,000 bus system was generated in this way and its scalability was tested using multiple processes.

The future work of this project involves using fieldsplit to apply different preconditioners in different sections of the multiphysics system.

References

[1] Sauer, P.W., and M. A. Pai. *Power System Dynamics and Stability*. New Jersey: Prentice Hall Inc., 1998.
[2] DMNetwork routines. Available online at: http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/DM/DMNETWORK.html
[3] PETSc Manual. Available online at: http://www.mcs.anl.gov/petsc/petsc-current/docs/manual.pdf